AFRL-IF-RS-TR-2001-103
Final Technical Report
May 2001

USING A LARGE CYC-BASED ONTOLOGY TO MODEL AND PREDICT VULNERABILITIES AT THE REAL-WORLD INFO-SYSTEM BOUNDARY

CYCORP, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. H504/00

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

AFRL-IF-RS-TR-2001-103 has been reviewed and is approved for publication.

[illegible]

APPROVED: PETER J. ROCCI, Jr.
Project Engineer

[illegible]

FOR THE DIRECTOR: JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

USING A LARGE CYC-BASED ONTOLOGY TO MODEL AND PREDICT VULNERABILITIES AT THE REAL-WORLD INFO-SYSTEM BOUNDARY

Blake Shepard

Contractor: CYCORP, Inc.
Contract Number: F30602-99-C-0142
Effective Date of Contract: 11 May 1999
Contract Expiration Date: 30 November 2000
Short Title of Work: Using a Large CYC_Based Ontology to Model and Predict Vulnerabilities at the Real-World Info-System Boundary
Period of Work Covered: May 99 – Nov 00

Principal Investigator: Blake Shepard
Phone: (512) 514-2952
AFRL Project Engineer: Peter J. Rocci, Jr.
Phone: (315) 330-4654

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Peter J. Rocci, Jr., AFRL/IFTD, 525 Brooks Rd, Rome, NY.

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE May 01	3. REPORT TYPE AND DATES COVERED Final May 99 - Nov 00	
4. TITLE AND SUBTITLE USING A LARGE CYC-BASED ONTOLOGY TO MODEL AND PREDICT VULNERABILITIES AT THE REAL-WORLD INFO-SYSTEM BOUNDARY			5. FUNDING NUMBERS C - F30602-99-C-0142 PE - 63760E PR - H504 TA - 34 WU - 01
6. AUTHOR(S) Blake Shepard			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) CYCORP, Inc. 3721 Executive Center Drive Austin, TX 78731			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFTD 525 Brooks Rd Rome NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-103
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Peter J. Rocci, IFTD, 315-330-4654			
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE
13. ABSTRACT *(Maximum 200 words)* This work is part of the new move toward content, as opposed to the architecture or methodology or algorithms, as the critical factor in complex information systems. The Information Assurance vulnerabilities problem is the quintessential example of "new and unexpected situations arising". In this effort, we have added a tremendous amount of knowledge to the Cyc Knowledge Base that enables Cyc to reason about cyber and non-cyber vulnerabilities, electronic attacks, and the relation between vulnerabilities and potential attacks.			
14. SUBJECT TERMS Information Assurance, Knowledge-Bases, Vulnerabilities			15. NUMBER OF PAGES 24
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Using a Large CYC®-Based Ontology To Model and Predict Vulnerabilities At the Real-World <--> Info-System Boundary

Cycorp
3721 Executive Center Drive, Suite 100
Austin, TX 78731

1. Introduction

Cycorp's work on IASET has focused primarily on two fronts. First, we have carefully and richly represented hundreds of concepts, constraints, and rules that enable Cyc to reason about the cyber and non-cyber vulnerabilities of information systems (approximately 600 constants and 4000 assertions). Second, we have developed mechanisms to enable Cyc to learn about an information system automatically and to provide a vulnerability assessment of it. Our twin goals were (1) to extend the existing Cyc-based HPKB IKB (Integrated Knowledge Base) ontology to cover intrusion, interception, battlefield, and related concepts that occur in patterns of real-world facts that should lead an INFOSEC officer or commander to suspect a possible vulnerability, and (2) to build a vulnerability analyst to trigger alerts and to provide a means to query about the vulnerability of a system. We have accomplished these goals. Cyc can take an automatically generated representation of a network in the manner just described and reason about its vulnerabilities using Cyc's full range of common-sense and cyber-vulnerability specific knowedge, to provide a sophisticated vulnerability assessment of the network.

In section 2 of this report, we will provide an overview of the Cyc technology. In section 3,we will describe the information we have added to the Cyc knowledge base about the cyber and non-cyber vulnerabilities of information systems. In section 4, we will described the technology we have developed to integrate Cyc with sources of information that enable it to learn about information systems automatically.

2. Overview of Cyc

Cyc technology consists of an immense multi-contextual knowledge base, an efficient inference engine, a set of interface tools, and a number of special-purpose application modules for Unix, Windows NT, Solaris and other platforms. The knowledge base is built on a core of over 1,000,000 hand-entered assertions designed to capture a large portion of what we normally consider to be "common-sense knowledge" about the world.

For example, Cyc knows that trees are usually outdoors, that once people die they stop buying things, and that glasses containing liquid should be carried rightside-up.

2.1. The Cyc knowledge base

The Cyc knowledge base ("KB") is an immense set of assertions about the world. Those assertions may be stated as expressions in CycL, the Cyc representation language, which is as expressive as first-oder logic with identity. The terms of CycL expressions can be variables, certain kinds of objects native to the computational substrate (such as strings or integers), or Cyc constants. Cyc contains objects in the KB which are created to denote particular concepts. Over the past fifteen years, approximately 100,000 concepts and just over 1 and 1/4 million rules and assertions that inter-relate them have been carefully hand-entered in the Cyc KB.

The Cyc objects which denote concepts are called "constants". They have unique names and are wri tten with the prefix "#$". Cyc constants can either denote collections, like "the collection of all electronic attacks", or individuals, like "the first electronic attack on our enclave last Thursday". Every term in Cyc is an element of #$Thing, the universal collection. #$Thing is partitioned into #$Individual and #$SetOrCollection. #$Individual denotes the set of all things which are not sets. Individuals in the Cyc KB include constants such as #$CityOfSanFrancisco, #$RonaldReagan, #$Internet, #$SIPRNet, and #$MicrosoftOfficeSuite-ComputerProgram. #$Collection denotes the set of all things that are not individuals. Collections in the KB include #$Person, #$VulnerabilityType, #$AttackByComputerOperation, and #$AttackOnObject.

The fundamental type of CycL expression is the "atomic formula", where a predicate is applied to one or more terms to indicate some relationship between the things denoted by the terms:

```
(<predicate> <term1> <term2> ...)
```

If there are no variables in the expression, all the terms are said to be "ground", and so the expression is referred to as a "ground atomic formula", or "GAF".

Predicates are all strongly typed, and a single collection must be specified as the type for each argument of every predicate.

For instance, the term #$performedBy, an instance of #$BinaryPredicate, has the following assertions:

```
(#$arg1Isa #$performedBy #$Action)
(#$arg2Isa #$performedBy #$Agent)
```

These assertions specify the domain and range of #$performedBy, the collection whose instances can be its first argument and the collection whose instances can be its second argument. So we can only use #$performedBy to specify some agent which performs some action.

Because every argument of every predicate in Cyc has type constraints, the space of valid assertions is radically reduced.

2.2. The Cyc inference engine

The Cyc inference engine handles modus ponens and modus tollens (contrapositive) inferencing, universal and existential quantification, and mathematical inferencing. It uses contexts called "microtheories" to optimize inferencing by restricting search domains.

The Cyc knowledge base contains hundreds of thousands of assertions. Many approaches commonly taken by other inference engines (such as frames, RETE, match, Prolog, etc.) just don't scale well to KBs of this size. As a result, the Cyc team has been forced to develop other techniques.

Cyc includes several hundred special-purpose inferencing modules for handling specialized common types of inference. One set of modules handles reasoning concerning collection membership, subsethood, and disjointness. Another handles equality reasoning. Still others implement symmetry, transitivity and reflexivity reasoning.

Backward inferencing--the type of inferencing initiated by an ASK operation--can be regarded as a search through a tree of nodes, where each node represents a CycL formula for which bindings are sought, and each link represents a transformation achieved by employing an assertion in the knowledge base.

For example, let's say I ask Cyc for bindings for the formula (#$likesObject ?x ?y). That formula will constitute the root node of an inference search tree. What I am looking for is any assertion which will help provide bindings for ?x and ?y. The KB may contain some some if-then rules, such as the default rule:

```
(#$implies
  (#$possesses ?x ?y)
  (#$likesObject ?x ?y))
```

This assertion would constitute a link to a new node with a different formula to satisfy, namely, the formula

```
(#$possesses ?x ?y)
```

Now Cyc might find an assertion in the KB that says

(#$possesses #$RonaldReagan #$ChocolateCandy002),

In which case Cyc would bind #$RonaldReagan to ?x and $ChocolateCandy002 to ?y.

The Cyc inference engine uses many specially-designed heuristic rules to decide which leaf node to expand next in an inference search. The heuristic rules are based on the synactic and semantic features of the formulas that occur at the nodes.

3. Representational work for IASET

Cycorp's IASET work is part of the new move toward *content*, as opposed to the architecture or methodology or algorithms, as the critical factor in complex information systems. The novel technology is in the meaning of the assertions in the knowledge base. There is a fundamental difference between brittle *ad hoc* expert systems, on the one hand, and a use-neutral knowledge base such as Cyc in which each piece of knowledge is entered at the highest appropriate level of generality such that new and unexpected situations that arise in the future can exploit that accumulated corpus of knowledge. The IA vulnerabilities problem is the quintessential example of "new and unexpected situations arising". If one could anticipate all dangers, one could addres them. In general, however, one can't anticipate them, so an INFOSEC officer or commander needs to be alerted to novel vulnerabilities as soon as they are suspected. Further, knowledge about the internal data state of a system will not, in general, suffice to ground vulnerability assessments. Many Red Force behaviors in the real world, and other external conditions, can indicate whether and when penetrations might occur (or may have occurred). Cyc is the only system that is sufficiently general and complete to reason effectively about novel suspected vulnerabilities and vulnerabilities that can be identified by looking at information external to the data world.

To prepare Cyc to provide novel and cokmplex vulnerability assessments, we needed to add a sophisticated representation of vulnerability to the Cyc knowledge base. For IASET, we have added a tremendous amount of knowledge to the Cyc KB that enables Cyc to reason about cyber and non-cyber vulnerabilities, electronic attacks, and the relation between vulnerabilities and potential attacks. In section 3.1, the vulnerability knowledge that has been added to Cyc is described. In 3.2, our approach to representing electronic attacks in Cyc is discussed. Finally, in 3.3, we describe the way we represent the relation between vulnerabilities and potential attacks.

3.1. The representation of vulnerabilities in Cyc

Representing the concept "vulnerability" is important for our project because understanding what it is to be vulnerable, how vulnerabilities interact, and how particualr vulnerabilities relate to particualr possible attacks is crucial for reasoning in a general way about novel possible threats.

English speakers use the term "vulnerability" in a variety of ways. Thus, there is a suite of predicates in CYC® to make assertions about vulnerability. Whenever something is vulnerable it has the potential to incur some sort of damage. We link vulnerability to an increased likelihood of incurring a particular type of damage rather than actual damage should the vulnerability conditions materialize. The reason for this is that a vulnerability itself, even in the conditions in which damage is likely to occur, is not, by itself, enough to conclude that damage does occur.

Our CycL representations of vulnerability fall into several intersecting camps. We distinguish "being vulnerable in a situation" from "being vulnerable to an object"; we have ways to express that an object's vulnerability has increased or decreased from some baseline; we can represent the fact that one situation (or object)makes an entity more vulnerable to a certain sort of damage than another situation (or object) does; and we have predicates for expressing an agent's vulnerability in virtue of some proposition being true (this last employs the full expressive power of Cyc); we can also represent the fact that vulnerabilities come in degrees. Finally, we sometimes want to say that something possesses a very common or well-known type of vulnerability, such as "vulnerability to the cold" or "Eric Allman Sendmail 8 vulnerability," and we have developed an efficient means of representing that sort of vulnerability.

What follows is a complete breakdown of the CycL predicates we use to represent vulnerabilities in Cyc.

3.1.1. "Vulnerability in a situation" versus "vulnerability to an object"

#$vulnerableIn and #$vulnerableTo are the CycL predicates we use to represent "vulnerability in a situation" and "vulnerability to an object", respectively. The purpose of #$vulnerableIn is to represent the vulnerabilities a thing has by virtue of playing a role in a situation. As an example of the distinction, consider a situation in which I am being assaulted by a mugger with a knfe. *In* that situation, I am vulnerable to being killed. Why? Because, in that situation, I am vulnerable *to* being mortally wounded by the knife

(#$vulnerableIn THREAT-SITUATION VULNERABLE-THING HARM-TYPE)

means that when in situation THREAT-SITUATION, VULNERABLE-THING is vulnerable to hardships of type HARM-TYPE. In contrast, the purpose of #$vulnerableTo is to represent the vulnerabilities one object has by virtue of the harmful capabilities of another object.

(#$vulnerableTo THREAT-OBJ OBJECT HARM-TYPE)

means that object THREAT-OBJ poses harm of type HARM-TYPE to OBJECT. In Cyc, we have represented the connection between being vulnerable in a situation and being vulnerable to an object as follows:

```
(#$implies
   (#$vulnerableIn ?SITUATION ?VULNERABLE-THING ?HARM-TYPE)
   (#$thereExists ?THREAT-OBJ
     (#$and
       (#$parts ?SITUATION ?THREAT-OBJ)
       (#$vulnerableTo ?THREAT-OBJ ?VULNERABLE-THING ?HARM-TYPE))))
```

This CycL rule represents the fact that whenever a thing is vulnerable to harm in a situation, there is some threatening object which is part of the situation and which is making the thing vulnerable to the harm. By virtue of this representation, if CYC® knew I was vulnerable to getting killed in a mugging event, e.g.,

```
(#$vulnerableIn #$Mugging001 #$Blake #$Killing-Biological)
```

Cyc would also be able to conclude that there is something in the event represented by #$Mugging001 that makes me vulnerable to being killed:

```
(#$thereExists ?THREAT-OBJ
     (#$and
       (#$parts #$Mugging001 ?THREAT-OBJ)
       (#$vulnerableTo  ?THREAT-OBJ #$Mugging001 #$Killing-Biological)))
```

3.1.2. Degrees of vulnerability

The predicate #$makesVulnerableToDegree enables Cyc to reason about changing degrees of vulnerability.

```
(#$makesVulnerableToDegree SITUATION OBJECT RESULT-TYPE DEGREE)
```

means that when in SITUATION, OBJECT is vulnerable to hardships of type RESULT-TYPE to degree DEGREE. We stipulate that the baseline level of vulnerability is #$Low, and that when sound security measures are in place, vulnerability is #$VeryLow. CycL rules, such as:

```
(#$implies
   (#$and
     (#$parts ?SIT3 ?SIT1)
     (#$parts ?SIT3 ?SIT2)
     (#$different ?SIT1 ?SIT2 ?SIT3)
     (#$makesVulnerableToDegree ?SIT1 ?OBJECT ?RESULT-SPEC #$Low)
     (#$makesVulnerableToDegree ?SIT2 ?OBJECT ?RESULT-SPEC #$Low))
  (#$makesVulnerableToDegree ?SIT3 ?OBJECT ?RESULT-SPEC #$Medium))
```

enable CYC® to conclude that when more than one concurrent situation makes an object vulnerable to the same ill effect to the same degree, vulnerability to that effect increases. By virtue of reasoning with this sort of rule, CYC® would know that if being sneezed on gave me a low-level vulnerability to catching a cold, then in a situatioin in which I was sneezed on many times, I'd be medium-level vulnerable to catching a cold.

3.1.3. Increasing and decreasing vulnerabilities

The CycL predicates #$increasesVulnerabilityIn, #$increasesVulnerabilityTo, #$decreasesVulnerabilityIn, and #$decreasesVulnerabilityTo (each of which is connected to #$makesVulnerableToDegree), are predicates that enable CycL representations of increasing and decreasing situational and object-related vulnerabilities.

```
(#$increasesVulnerabilityIn SITUATION OBJECT RESULT-TYPE)
```

means that being in SITUATION increases OBJECT's vulnerability to RESULT-TYPE. This predicate is similar to #$makesVulnerableToDegree except that, unlike #$makesVulnerableToDegree, it expresses a relative increase in OBJECT's vulnerability rather than an absolute degree of vulnerability. #$increasesVulnerabilityIn is connected to #$makesVulnerableToDegree via the following rule:

```
(#$implies
    (#$and
      (#$startsAfterEndingOf ?SIT2 ?SIT1)
      (#$greaterThan ?DEG2 ?DEG1)
      (#$makesVulnerableToDegree ?SIT2 ?OBJ ?HARM ?DEG2)
      (#$makesVulnerableToDegree ?SIT1 ?OBJ ?HARM?DEG1))
    (#$increasesVulnerabilityIn ?SIT2 ?OBJ ?HARM))
```

This rule says that if one situation follows another, and the degree of vulnerability of an object in both situations moves from a relatively low value to a relatively high value, the object's vulnerability increases.

3.1.4. Common types of vulnerability

We have created specific CycL contants to denote particular common types of vulnerability. #$Vulnerability the most general type of vulnerability. #$Vulnerability is the collection of #$StaticSituations in which the salient focal relationship which does not change is that between a vulnerable entity and that which makes the entity vulnerable. We use the CycL predicate #$hasVulnerabilityType to represent in Cyc that an object has a particular type of vulnerability. For example, the assertion:

```
(#$hasVulnerabilityType #$UserAccount001 #$UnauthorizedLoginVulnerability)
```

means that #$UserAccount001 is vulnerable to unauthorized logins. Here is a representative list of the vulnerability types that have been represented in Cyc:

#$BufferVulnerability
#$CanonicalPasswordVulnerability
#$ComputerVirusVulnerability
#$CyberInfiltrationVulnerability
#$CyberVulnerability
#$CyberVulnerabilityExploit
#$DenialOfServiceVulnerability
#$DictionaryPasswordVulnerability
#$ExecutableStackVulnerability
#$FilePermissionVulnerability
#$InsecureAccountVulnerability
#$InsecureInformationVulnerability
#$InsecurePasswordVulnerability
#$InsecureSoftwareVulnerability
#$JoePasswordVulnerability
#$NoteAboutVulnerability
#$PhysicalDamageVulnerability
#$PhysicalInsecurityVulnerability
#$PhysicalVulnerability
#$PlaintextPasswordStorageVulnerability
#$PlaintextPasswordVulnerability
#$SGIDVulnerability
#$SUIDRootVulnerability
#$UnauthorizedLoginVulnerability

We represent sufficient conditions for all such common or well-known types of vulnerability in Cyc. We do this by writing CycL rules that, in their antecedents, specify the sufficient conditions for the reified vulnerabilitiy types mentioned in their consequents. For example, the rule:

```
(#$implies
    (#$and
      (#$isa ?PASSWORD #$Password-Weak)
      (#$passwordForUnixAccount ?AGENT ?ACCOUNT ?PASSWORD))
    (#$hasVulnerabilityType ?ACCOUNT #$UnauthorizedLoginVulnerability))
```

says that being a Unix account with a weak password is sufficient for having an unauthorized login vulnerability (#$Password-Weak denotes the collection of all canonical, short, and lexical passwords).

3.1.5. Sources of Vulnerability Data

A great deal of the knowledge we represented about cyber vulnerabilities for IASET was accessed from various online sources of cyber vulnerability data. The online sources were vulnerability databases or archived computer security mailing lists. Online vulnerability databases, such as the one at www.securityfocus.com, are typically arranged so that the specific vulnerabilities of computer programs are indexed by the operating system on which they are known to cause problems. For example, Eric Allman sendmail version 8.x is known to have a specific vulnerability for Linux, and we represented that information in the Cyc KB for IASET. The archived vulnerability mailing lists are less structured, but still proved to be an excellent source of representable vulnerability information.

3.2. The representation of electronic attacks in CYC®

What in English are called "cyber attacks" or "electronic attacks" are represented in Cyc by constants that denote subcollections of #$AttackByComputerOperation, which istelf is a specialization of #$AttackTypeByWeaponType: it is the collection of all attacks executed using computer operations as weapons.

Prior to the commencement of our IASET work, Cyc already knew a significant amount about attacks in general, and that knowledge is inherited by the representations of electronic attacks for IASET. For example, Cyc knows:

```
(#$implies
    (#$and
      (#$isa ?ATT #$AttackOnObject)
      (#$successfulForAgents ?ATT ?DOER)
      (#$objectAttacked ?ATT ?OBJ))
    (#$damages ?ATT ?OBJ)),
```

which means that a successful attack on an object damages it. CYC® also knows:

```
(#$implies
   (#$and
     (#$isa ?ATT #$AttackOnObject)
     (#$performedBy ?ATT ?DOER)
     (#$objectAttacked ?ATT ?OBJ))
   (#$purposeInEvent ?DOER ?ATT
     (#$damages ?ATT ?OBJ)))
```

which means that those who perform attacks do so with the intent of damaging the objects they attack.

The vocabulary used to name constants in the hierarchy of CycL constants that represent cyber attacks usually has the format '#$ElectronicAttack-x', where 'x' refers to a conventional name of an attack (such as 'denial of sevice'). Each of these collections is a subcollection of #$AttackByComputerOperation, and each is an instance of some type level collection in the electronic attack hierarchy. Some constants in the electronic attack hierarchy have the format "#$ElectronicIntelligenceAttack-x". These constants designate subcollections of #$ElectronicIntelligenceAttack-General, which itself is a subcollection of #$AttackByComputerOperation, and an instance of #$ElectronicAttackType.

For IASET, we have reified numerous subcollections of #$AttackByComputerOperation. Here is a representative list:

#$ElectronicAttack-Bonk
#$ElectronicAttack-BufferOverflow
#$ElectronicAttack-ComputerCrashing
#$ElectronicAttack-Coordinated
#$ElectronicAttack-CorruptionOfInformation
#$ElectronicAttack-DataFlooding
#$ElectronicAttack-DefacingAWebsite
#$ElectronicAttack-DenialOfService
#$ElectronicAttack-DestructionOfInformation
#$ElectronicAttack-Distributed
#$ElectronicAttack-EMailBomb
#$ElectronicAttack-LogicBomb
#$ElectronicAttack-Smurfing
#$ElectronicAttack-SynFlooding
#$ElectronicAttack-TearDrop
#$ElectronicAttack-TimeBomb
#$ElectronicAttack-UDPPacketStorm
#$ElectronicAttack-Vandalism
#$ElectronicAttack-Virus
#$ElectronicAttack-Worm

The subcollections of #$AttackByComputerOperation are richly interconnected. For example, #$ElectronicAttack-DefacingAWebsite is a more specific subcollection of #$ElectronicAttack-Vandalism. #$ElectronicAttack-DataFlooding, #$ElectronicAttack-EMailBomb, and #$ElectronicAttack-ComputerCrashing are all subcollections of #$ElectronicAttack-DenialOfService. #$ElectronicAttack-Bonk, #$ElectronicAttack-SynFlooding, #$ElectronicAttack-TearDrop, and #$ElectronicAttack-UDPPacketStorm are all subcollections of both #$ElectronicAttack-ComputerCrashing and #$ElectronicAttack-DataFlooding.

We have represented in Cyc elaborate specific information about each type of electronic attack. For example, the rule

```
(#$implies
   (#$and
     (#$objectActedOn ?EVNT ?OBJ)
     (#$isa ?EVNT #$ElectronicAttack-DenialOfService))
   (#$thereExists ?ACT
     (#$and
       (#$holdsIn (#$STIB ?EVNT)
         (#$behaviorCapable ?OBJ ?ACT #$deviceUsed))
       (#$holdsIn (#$STIF ?EVNT)
         (#$not
           (#$behaviorCapable ?OBJ ?ACT #$deviceUsed))))))
```

says that objects that are the targets of successful denial of service attacks are capable of functioning before they are attacked, but not after they are attacked. Also, the rule:

```
(#$implies
   (#$isa ?SYNFLOOD #$ElectronicAttack-SynFlooding)
   (#$likelihood
     (#$isa ?SYNFLOOD #$ElectronicAttack-Distributed) #$HighToVeryHigh))
```

says that syn flooding attacks are very likely to be distributed attacks. There are dozens of specific rules like these for all the subcollections of #$AttackByComputerOperation.

3.3. Linking vulnerabilities to potential attacks

Cyc understands the connection between vulnerabilities and potential attacks. One of the most efficient ways Cyc can reason about the relation between vulnerabilities and potential attacks is with what we call a "script-oriented" approach.

The script-oriented approach starts with the representation in CycL of common or well-known types of vulnerability. We create a constant that represents a type of situation in which an entity is vulnerable to a certain type of ill-effect. We link these representations of common and well-known vulnerabilities to other constants in the Cyc KB by specifying the sufficient conditions for having them. For instance, we represent "network node access vulnerability" and we say that a network with nodes accessible to unauthorized agents has a network node access vulnerability:

```
(#$implies
   (#$and
      (#$isa ?NETWORK #$LocalAreaNetwork)
       (#$nodeInSystem ?HUB ?NETWORK)
```

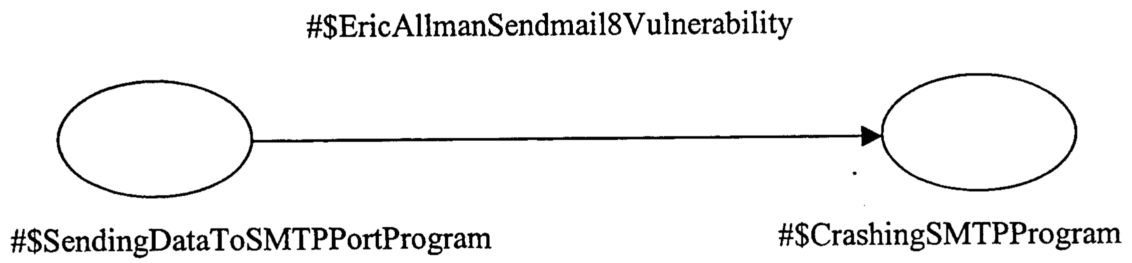
```
            (#$hasPhysicalAccess ?AGENT ?HUB)
            (#$unauthorizedAgent ?AGENT ?HUB))
        (#$hasVulnerabilityType ?NETWORK #$NetworkNodeAccessVulnerability))
```

The next step in doing script-based vulnerability assessment in Cyc is to represent the conditions, including types of common vulnerabilities, that enable one to move from the performance of one action type to another.

Here's an example of the way this works in Cyc . We represent the fact that the vulnerability represented by the CycL constant #$EricAllmanSendmail8Vulnerability enables one to move from sending data of a particular sort to the SMTP port, to crashing the SMTP program:

```
 (#$actionTypeAllowsActionTypeWhen
      #$SendingDataToSMTPPortProgram
      #$CrashingSMTPProgram
      #$EricAllmanSendmail8Vulnerability)
```

What this assertion says could be represented as a directed graph in which the action types are nodes and the condition connecting the nodes is a directed link:


#$EricAllmanSendmail8Vulnerability
#$SendingDataToSMTPPortProgram
#$CrashingSMTPProgram

By representing all significant condition types that are sufficient for significant action-types, we will end up with an elaborate script that represents all possible paths from start actions to goal actions. There are currently 133 instances of #$CycSystemPathConstant (such as #$n-wayJunctionInSystem, #$sourceNodeInSystem, #$cutNodeInSystem, etc.), which are richly interconnected with rules that enable Cyc to reason deeply about the relations of nodes and links in a script.

The next step in performing script-based assessments is determining which conditions and vulnerabilities obtain on the network being assessed. So, if it turns out that a system is running software that would enable one to move from a performance of one action type to the performance of another action type, Cyc knows it. Finally, it is possible to ask Cyc which attack paths in the script can be performed on the network being assessed.

So, if we specify the start node as the action type "Scanning a network", it may turn out that, acting on a particular network represented in Cyc, there are a few paths through the script that leads to the goal electronic attack action type "Gaining root access".

4. Automatic semantic integration of security-relevant knowledge

We have enabled automatic semantic integration of a number of security-relevant knowledge sources with Cyc. By "semantically integrated", I mean that we take the output of each of these sources, automatically encode it in CycL, and assert it in the knowledge base.

Currently, we have four automatic means of integrating specific network information with the Cyc knowledge base. traceroute provides a logical network topology, nmap provides information about what ports are open, queso identifies the OS each machine on a network is running, and by secure shelling to each machine on a network, we can automatically look at relevant files and run scripts to pick up MAC addresses, CPU speed, and RAM.

From the command line, we run a single program that executes all these functions on a network, converts their output to CycL assertions, and incorporates these assertions in the general Cyc knowledge base.

Cyc can take the representation of a network that has been automatically generated in the manner I just described and reason about its vulnerabilities using Cyc's full range of common-sense and cyber-vulnerability specific knowedge, to provide a sophisticated vulnerability assessment of the network.

5. Future Directions

The IASET work done at Cycorp has, we believe, positioned Cycorp to develop a powerful commercial network vulnerability assessment tool. The tool we develop will be significantly different than existing network vulnerability assessment tools. Instead of discovering vulnerabilities by attempting expoits, the tool we develop will represent a network declaratively and utilize Cyc's inference engine and vulnerability knowledge to deductively determine what vulnerabilities a network has. Instead of being limited to reasoning about cyber vulnerabilities, the tool we develop will be able to reason about any sort of vulnerability, drawing on Cyc's broad common-sense real-world knowledge. Also, our tool will not be limited to performing vulnerability assesment. It can become a general-purpose network administration tool. A system administrator will be able use our tool's declarative representation of a network to test the impact of *any* sort of change to the network before implementing those changes. Also, our tool will be a general risk-management tool, that automates reasoning about the sorts of security risks it is acceptable to take given certain assumptions about the intended functionality of a network.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.